Hindawi
Shock and Vibration
Volume 2021, Article ID 8847664, 7 pages
https://doi.org/10.1155/2021/8847664

Research Article

A Coupling Graphic Pipeline with Normal Mode Model for Rapid Calculation of Underwater Acoustic Field

Jianghao Zhuo, Ling Wang, Ke Xu, and Jianwei Wan

College of Electronic Science, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Jianwei Wan; kermitwjw@139.com

Received 7 September 2020; Revised 27 December 2020; Accepted 18 January 2021; Published 29 January 2021

Academic Editor: Erkan Oterkus

Rapid execution is required in operation-oriented applications in underwater acoustic modelling. In this paper, the GPU graphic pipeline is used to accelerate the calculation of high-resolution sound field image in the normal mode model of underwater acoustic propagation. The computer times of the proposed graphic pipeline method, the MATLAB code, and the C# code are compared for a stratified shallow water waveguide using the KRAKEN model at different frequencies. The research validates that the graphic pipeline method outperforms the classic CPU-based methods in terms of execution speed at the frequencies where the eigenvalue equation in normal mode models can be solved.

1. Introduction

Underwater acoustics entails the development and employment of acoustical methods to image underwater features, to communicate information via the oceanic waveguide, or to measure oceanic properties [1]. Sound waves are the main carrier for long-distance transmission of information underwater, so it is necessary to understand how they propagate in oceanic media [2]. Underwater acoustic models analytically or numerically predict the propagation of sound waves in the ocean by translating our physical understanding of sound into mathematical formulas solvable by computers. The primary motivation for the research of underwater acoustic models is related to antisubmarine warfare, and now, they are routinely used in both civil and military applications [3, 4].

According to application scenarios, modelling applications in underwater acoustics generally fall into one of the two categories: research-oriented or operation-oriented [1]. Research-oriented applications are performed in laboratory conditions where the available computing platform is powerful, so accuracy is critical and computer time is not important, such as the design of sonar systems. To ensure accuracy, input parameters of these applications are usually detailed, so the marine environment needs to be finely measured. However, operation-oriented applications are executed in demanding conditions where computing power is limited and a quick decision to the current situation is required, such as fleet operations or underwater acoustic communication. Therefore, the challenge in operation-oriented applications is to achieve high processing speed on low-performance computers. In addition, it is usually not feasible to perform fine measurement of the marine environment in demanding conditions, so the input parameters of operation-oriented applications are not as detailed as research-oriented applications.

Underwater acoustic modelling has been developed for decades [13, 14]. One of the most widely used models is based on normal mode theory. An early normal mode model is a simple two-layer model proposed by Pekeris [5]. Initially, normal mode theory assumes range independence that the oceanic parameters (sound speed, bathymetry, etc.) change with depth only. To extend to range dependence that oceanic parameters change with both depth and distance, distance segment and mode coupling are applied [6]. The output of normal mode models is usually a sound field image (sound field in a distance-depth grid) to intuitively show the underwater sound field varying with distance and depth. This intuitive representation of the underwater sound field is essential in operation-oriented applications. However, when the resolution of the underwater sound field image is high, calculating sound field may take up a lot of computing resources. In [7], the computer times of two different numerical solutions for normal mode theory are tested using a one-thread CPU on Tianhe-2 supercomputer, and for both numerical solutions, the times to calculate sound field image are much longer than the times to calculate the modes. For operation-oriented applications, the acceleration in calculating sound field image is necessary.

With the development of GPU hardware and graphic interface, it is feasible to transfer large-scale parallel computation from CPU to GPU, either by general purpose computation on GPU (GPGPU) or by adopting graphic pipeline for specific computation [15]. The purpose of graphic pipeline is to generate or render a 2D image on the screen from a 3D scene, and displaying a sound field image on the screen is a subset of this purpose. Since OpenGL 2.0, OpenGL ES 2.0, or DirectX 8.0, GPUs start to support programmable graphic pipelines. Programmable graphic pipelines provide developers an interface to customize the 2D texture output by the GPU by programming in the geometry stage and rasterizer stage, such as implementing lighting models, shadows, and transparency. Since calculation of sound field images is repetitive and parallel, in this paper, calculating sound field in a distance-depth grid is transferred to the fragment shader in the rasterizer stage, and the parallel calculation of the sound pressure is realized by the screen postprocessing operation. Three implementations are compared in terms of computer times in this paper: the proposed graphic pipeline program of high-level shading language (HLSL) code, the MATLAB code based on vector operations, and the C# code using a one-thread CPU.

2. Materials and Methods

2.1. Normal Mode Model. Modelling underwater acoustic propagation generally starts with the wave equation or Helmholtz equation. Directly solving the time-varying wave equation produces time-domain methods, such as finite difference method (FDM), finite element method (FEM), and boundary element method (BEM), while solving the Helmholtz equation assuming that density and sound speed of sea water are independent of time produces frequency-domain method, such as normal mode method (NM), parabolic equation (PE) method, and ray theory (RT) method [9]. The 2D Helmholtz equation in a stratified medium with depth z and horizontal distance r is as follows:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial p}{\partial r}\right)+\rho(z)\frac{\partial}{\partial z}\left(\frac{1}{\rho(z)}\frac{\partial p}{\partial z}\right)+\frac{\omega^2}{c^2(z)}p=-\frac{\delta(r)\delta(z-z_s)}{2\pi r}, \tag{1}$$

where $\rho(z)$ and $c(z)$ are the density and sound speed of sea water, respectively, z_s is the source depth, ω is the angular frequency, and $p(r,z)$ is the sound pressure.

In normal mode theory, sound pressure is decomposed as $p(r,z)=R(r)Z(z)$ using the separation variable method to find a homogeneous solution of equation (1). The function $Z(z)$ satisfies

$$\begin{cases}\rho(z)\dfrac{\mathrm{d}}{\mathrm{d}z}\left(\dfrac{1}{\rho(z)}\dfrac{\mathrm{d}Z(z)}{\mathrm{d}z}\right)+\left(\dfrac{\omega^2}{c^2(z)}-k_r^2\right)Z(z)=0,\\[2ex] \dfrac{\mathrm{d}Z(0)}{\mathrm{d}z}=\mu_0\rho(0)Z(0),\\[2ex] \dfrac{\mathrm{d}Z(D)}{\mathrm{d}z}=\mu_D\rho(D)Z(D),\end{cases} \tag{2}$$

where D is the bathymetry, k_r^2 is a separation constant in solving the homogeneous equation, and μ_0 and μ_D are two constants that define boundary conditions at sea surface and sea bottom. Equation (2) is a Sturm–Liouville eigenvalue problem. Pairs of eigen functions and eigenvalues $(Z_m(z), k_{rm}^2)$, $m=1,2,\ldots$ can be solved from equation (2) if sufficient boundary conditions at sea surface and sea bottom are given. The sound pressure in the normal mode theory is as follows:

$$p(r,z)=\frac{i}{4\rho(z_s)}\sum_{m=1}^{\infty}Z_m(z_s)Z_m(z)H_0^{(1)}(k_{rm}r), \tag{3}$$

where $H_0^{(1)}(\cdot)$ is the first-type Hankel function.

Using the asymptotic approximation of Hankel function, the sound pressure in equation (3) is as follows:

$$p(r,z)\approx\frac{i}{\rho(z_s)\sqrt{8\pi r}}e^{-(i\pi/4)}\sum_{m=1}^{\infty}Z_m(z_s)Z_m(z)\frac{e^{ik_{rm}r}}{\sqrt{k_{rm}}}. \tag{4}$$

A sound field image is defined as the sound pressure in equation (4) in a distance-depth grid as $r=r_{\min}, r_{\min}+\Delta r, r_{\min}+2\Delta r,\ldots,r_{\max}$ and $z=z_{\min}, z_{\min}+\Delta\ z,\ z_{\min}+2\Delta z,\ldots,z_{\max}$.

2.2. Graphic Pipeline Treatment of the Normal Mode Model. Modern graphic pipeline usually includes three stages: application stage, geometry stage, and rasterizer stage [8]. The application stage is implemented by CPU, and developers have absolute control over this stage. In this stage, rendering data, including but not limited to vertex data, textures, and buffers, are prepared and transferred to GPU. The geometry stage performs per-vertex or per-polygon operations, and the transformation of vertex positions from world coordinates to screen coordinates is completed at this stage. The main task of the rasterizer stage is to determine the color of the pixels in each rendering primitive (usually a triangle), and how these pixels should be drawn on the screen (such as removing occluded pixels in the depth test or achieving transparency in the blending operation). Triangle setting and triangle traversal in the rasterization stage interpolate the output registers of each vertex in the geometry stage, and the interpolated registers are the input registers of the fragment shader corresponding to each pixel on the rendering primitive. Pixel colors are calculated in parallel in the fragment shader.

Details of these stages are shown in Figure 1. The colors indicate the configurability and programmability of different stages. Green indicates that the stage is programmable,

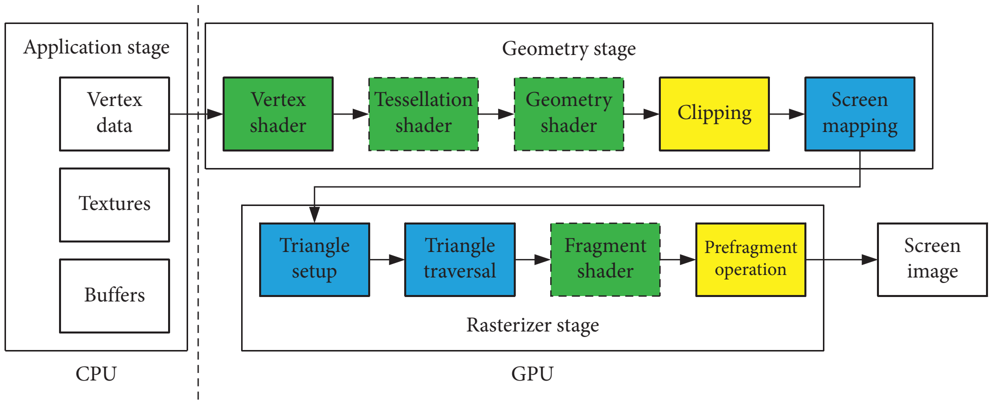

Figure 1: Implementation for graphic pipeline.

yellow indicates that the stage is configurable but not programmable, and blue indicates that the stage is fixedly implemented by the GPU. For programmable stages, the solid line indicates that the stage must be programmed by the developer, and the dotted line indicates that the stage is optional.

Figure 2 intuitively shows the process of rendering a triangle on the screen. Three vertices of the triangle are transformed from world space to screen space in the geometry stage. Then, the triangle in screen space is rasterized and represented by a series of fragments. Finally, the color of each fragment is calculated in parallel by fragment shader. Note that DirectX defines the upper left corner of the screen as the origin of the screen space, while the origin in OpenGL is at the lower left corner.

In this paper, the gridding in distance r and depth z of the sound field in equation (4) is mapped to the rasterization in the screen space. Settings of graphic pipeline to implement the mapping are shown in Figure 3. Geometry setup on the screen is shown in Figure 3(a). The image is represented by two connected triangles that cover the screen. The four vertices are located on the four edges of the view frustum in world space. Figure 3(b) shows the texture coordinates (2D vectors also known as UVs) as the output from vertex shader. In the triangle traversal, output registers of vertex shader are interpolated as the input of fragment shader. Supposing that the screen resolution is $W \times H$ and (x, y) is the center position of the pixel, the input UV of the fragment shader is $((x/W), (y/H))$.

The mapping from the input texture coordinate (u, v) of a fragment shader to its grid (r, z) is as follows:

$$(r, z) = \left(r_{\min} + u \times \left(r_{\max} - r_{\min}\right), z_{\min} + v \times \left(z_{\max} - z_{\min}\right)\right), \tag{5}$$

where $(r_{\min}, z_{\min})$ and $(r_{\max}, z_{\max})$ define the area of sound field image.

The flowchart of fragment shader is shown in Figure 4. The input of fragment shader is the interpolated UV and the output is the pixel color representing the sound pressure. Color mapping is flexibly designed according to different applications. Normal mode solutions (pairs of eigen functions and eigenvalues of equation (2)) are transferred from RAM (memory directly accessible by CPU) to VRAM (memory directly accessible by GPU) through buffer or texture, depending on the shader model supported by the GPU.

3. Results and Discussion

To test the computer time of calculating sound field image in the graphic pipeline, the case of a stratified shallow water waveguide is performed. The sound speed profile and density are shown in Figure 5. In the test case, the surface is assumed to be a pressure release boundary and the bottom is assumed to be a rigid boundary, that is, $\mu_0 = \infty$ and $\mu_D = 0$ in equation (2). The sound source depth is 30 m.

A general numerical method for solving the eigenvalue problem of equation (2) of arbitrary boundary conditions is FDM. In FDM, the sea water is divided into N equally spaced grids in depth. By replacing differential operators with difference operators, equation (2) is transformed into computer-solvable discrete algebraic equations. One of the widely used FDM is the KRAKEN model [10], where equation (2) is transformed into the following matrix eigenvalue equation:

$$AZ = h^2 k_r^2 Z, \tag{6}$$

where h is the grid width (in the test case, it is 10% of the wave length), A is an N-order tridiagonal matrix, and k_r^2 and Z are defined in equation (2).

Solving equation (6) is a mathematical problem. In the test case, equation (6) is solved by the function “eig (A)” in MATLAB, and the eigenvalues and eigenvectors are stored as a binary file as the input of the sound field image module.

A simple way to implement the graphic pipeline in Figure 3(a) is to use screen postprocessing. Screen postprocessing is a technique commonly used in game development and film production, and it has been robustly implemented in various commercial game engines. In the test case, Unity 5 (version 2018.3.11f1, 64 bit) is adopted to develop the screen postprocessing program. Guidelines of developing screen postprocessing in Unity 5 are accessible in [11], and a simple framework can be accessed by [12].

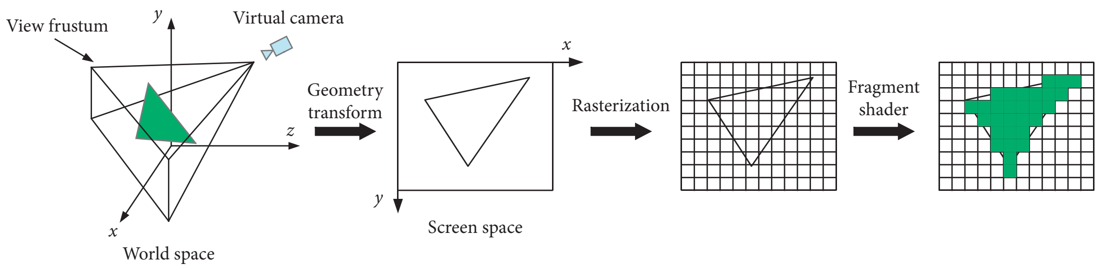

Figure 2: The process of rendering a triangle on the screen.

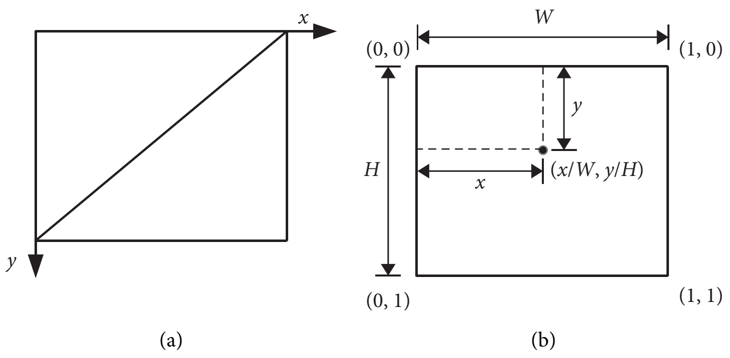

Figure 3: Settings of graphic pipeline. (a) Geometry setup of sound field image on the screen. (b) UVs of the four vertices.

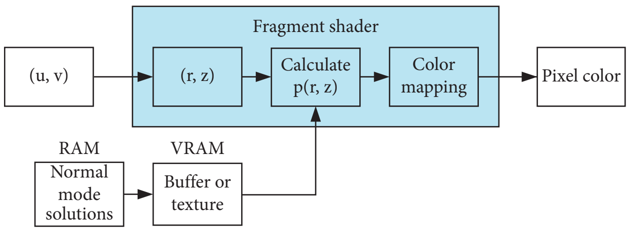

Figure 4: The flowchart of fragment shader.

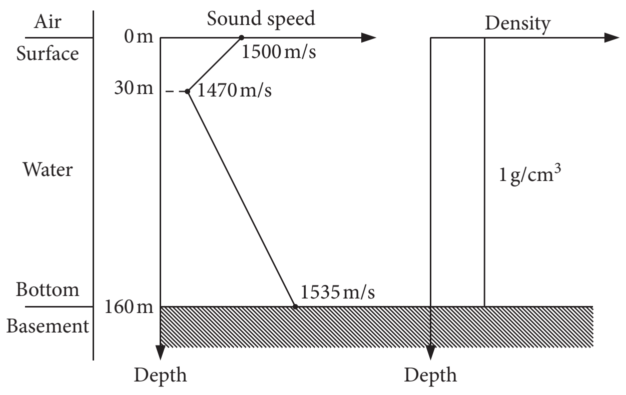

Figure 5: Sound speed profile and density in the test case.

In operation-oriented applications, transmission loss (TL) is used more generally than sound pressure. Both the sonar equation and the figure of merit (FOM) are measured by TL. TL is defined as

$$\mathrm{TL} = 20\log_{10}\left(\frac{|p(r,z)|}{|p_0(r=1)|}\right), \tag{7}$$

where $|p_0(r)| = 1/(4\pi r)$.

The first step of color mapping is to calculate TL by equation (7). Then, TL is mapped to the brightness (B) in HSB color space. For example, linear mapping and scaled mapping are shown in Figure 6.

Sound field images of the test case in Figure 5 are obtained by the MATLAB code and the screen postprocessing program (HLSL code), respectively. The frequency of the sound source is 200 Hz. The scope in depth is from 0 m to 160 m. The scope in distance is from 10 m to 300 m. The resolution is 1366×768. The sound field image calculated by the MATLAB code (MATLAB 2015a) is shown in Figure 7. The screenshot of the HLSL code is shown in Figure 8. To compare the result of the HLSL code with that of MATLAB, color mapping of the HLSL code is consistent with the color bar of MATLAB.

Intuitively, the image in Figure 8 is essentially the same with that in Figure 7. Although there is no obvious difference between Figures 7 and 8, the difference between MATLAB and graphic pipeline may still cause a slight difference between the two images. One of the most important differences is the word length. The word length on MATLAB is 64 (double precision), while in the graphic pipeline, the word length is 32 (single precision). To numerically compare Figures 7 and 8, it is necessary to obtain the TL calculated in the fragment shader. A feasible method is to convert the render texture maintained by the screen postprocessing into a 2D texture in the format of RGBA32, and the RGBA channels represent TL. The integer part of TL is represented

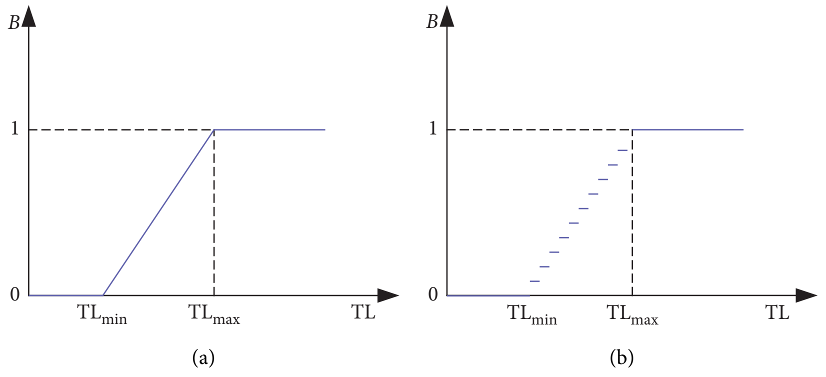

Figure 6: Examples of color mapping. (a) Linear mapping. (b) Scaled mapping.

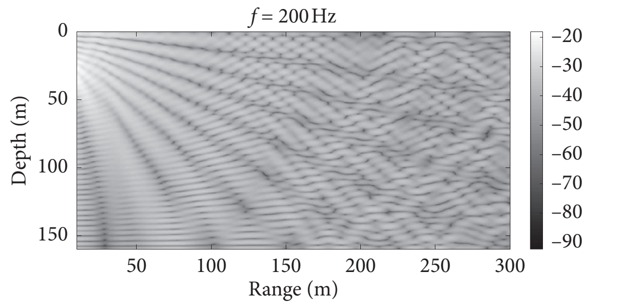

Figure 7: Sound field image by MATLAB code. The color bar is "gray" in MATLAB.

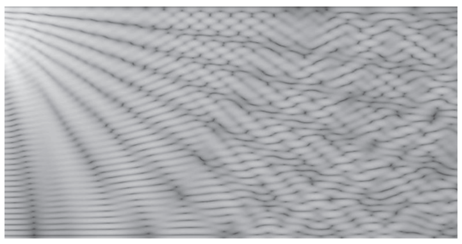

Figure 8: Screenshot of the HLSL code. A scaled mapping of 64 color scales is used, and the maximum and minimum values of TL are −18.0972 dB and −92.1202 dB.

by the R channel, and the decimal part is represented by the G, B, and A channels. The curves of TL at z = 30m by the MATLAB code and the HLSL code are shown in Figure 9, and the Taylor diagram [16] of the HLSL code and MATLAB code is shown in Figure 10. In Figure 10, the points of these two codes are close. From Figures 7–10, it is concluded that the results of HLSL code are correct and reliable both intuitively and numerically.

Table 1 shows the computer times of the MATLAB code and the HLSL code. To exclude that the difference in computer times may be caused by platform differences, the computer time of the C# code running in the main thread (one-thread CPU) on the Unity 5 platform is also shown in Table 1. Due to the long running time of the C# code, image resolution in the C# code is 683×3 (1/512 of 1366×768 resolution), and the computer time is multiplied by 512 to convert to that of 1366×768 resolution. The CPU is Intel Core i5-5200U, and the GPU is NVIDIA GeForce 920M. The result of MATLAB code is the average time of ten repeated runs. The HLSL code and C# code are executed once per frame, and their computer times are measured by their frame counts over a long period of time, as follows:

$$T_{\text{code}} = \frac{T_0}{N_{\text{code}}} - \frac{T_1}{N_{\text{ambient}}}, \tag{8}$$

where T_{code} is the computer time, N_{code} is the frame count in T_0 when HLSL code or C# code is running, and N_{ambient} is the frame count in T_1 when neither code is running.

As shown in Table 1, HLSL code has a great improvement in computational efficiency compared to MATLAB code and C# code. However, it may still be an accidental result caused by specific CPU and GPU. Moreover, the number of normal modes that depend on the acoustic frequency has an influence on the amount of calculation. To make the test general, the three codes are run on four computers over acoustic frequencies from 100 Hz (107 modes) to 100 kHz (106667 modes). Table 2 shows the CPU and GPU models of the four computers. Figure 11 shows the computer times of HLSL code, MATLAB code, and C# code on the four computers. Figure 12 shows the HLSL gain over MATLAB code or C# code (defined by the ratio of computer times).

It is shown in Figure 11 that for each computer, the calculation time of the HLSL code is significantly lower than that of the MATLAB code and C# code. It is worth noting that even HLSL code running on a low-performance GPU (that of computer 1) has a shorter computer time than that of MATLAB code running on a high-performance CPU (that of computer 4). In Figure 12, below 10 kHz, HLSL gain over MATLAB code decreases as the number of modes increases, and above 10 kHz, it tends to be a constant. In addition, HLSL gain over C# code keeps high in the entire test frequency band. Moreover, when the frequency is at 100 Hz, HLSL gain over MATLAB code even exceeds 1000 on computer 3 and computer 4. Therefore, it is reasonable to conclude that the HLSL code is much faster than the MATLAB code and the C# code in calculating the sound field image in normal mode models.

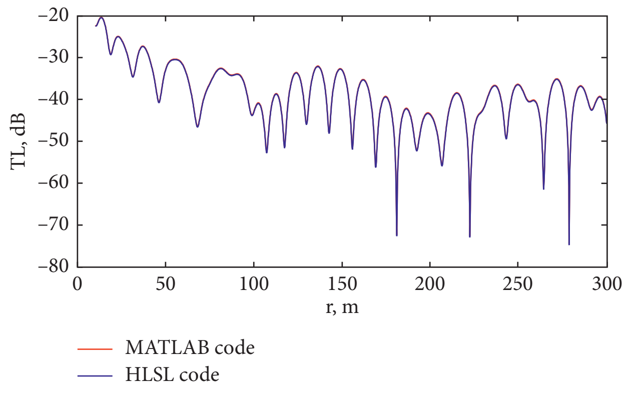

Figure 9: TL when the depth is 30 m.

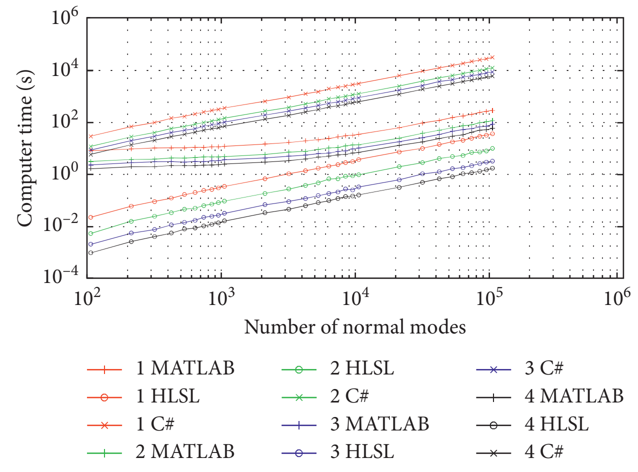

Figure 11: Computer times versus the number of normal modes.

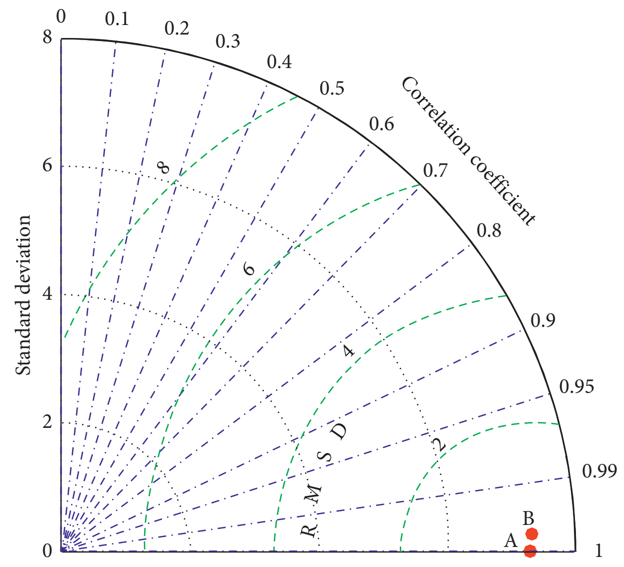

Figure 10: Taylor diagram of the HLSL code (point B) and MATLAB code (point A). The correlation coefficient is 0.9993, and the root mean square is 0.2811.

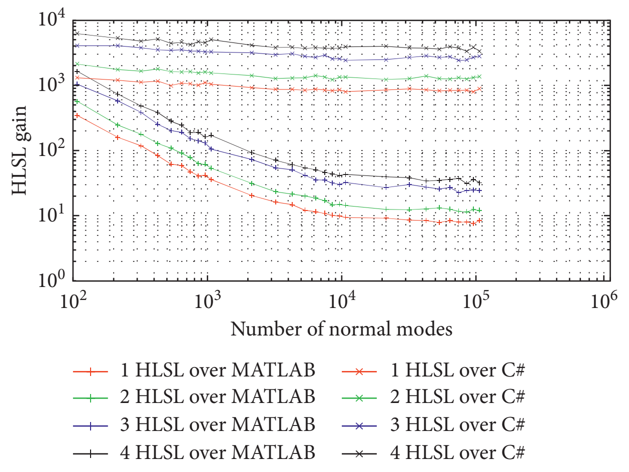

Figure 12: HLSL gain over MATLAB code or C# code on four computers.

Table 1: Computer times of MATLAB code, HLSL code, and C# code.

Method	Computer time (s)
MATLAB code	9.5578576
HLSL code	0.058186365
C# code	69.41085745

Table 2: CPU and GPU models of the four computers.

Computer	CPU	GPU
1	Intel Core i5-5200U	NVIDIA GeForce 920M
2	AMD Ryzen 5 2500U	NVIDIA GeForce GTX 960M
3	Intel Core i7-8706G	NVIDIA GeForce GTX 1060M
4	Intel Xeon E-2286M	NVIDIA GeForce RTX 2080

4. Conclusions

In this paper, the calculation of the sound field image in the normal mode model is implemented by the fragment shader in GPU graphic pipeline. The calculation is accelerated because the fragment shader is calculated in parallel. In a test case of a stratified shallow water waveguide, the computer times of the GPU-based HLSL code, the CPU-based MATLAB code, and C# code are compared. The results show that the computer time of the HLSL code is much shorter than those of the other two codes at the frequencies where the eigenvalue equation in normal mode models can be solved.

Although the method proposed in this paper is based on the normal mode model, it provides a possible acceleration method to the general problem set of calculating the sound field image in a 2D grid. For example, in the ray method and parabolic equation method, the simulation results are also represented by the sound field image on the depth-distance grid in general cases.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the "Double First-Class" Construction Guidance Project of the National University of Defense Technology (4345161111L).

References

[1] P. C. Etter, *Underwater Acoustic Modeling and Simulation*, CRC Press, Boca Raton, FL, USA, 5th edition, 2018.

[2] L. Bosheng and J. Lei, *Principles of Underwater Acoustics*, Harbin Engineering University Press, Harbin, China, 2010.

[3] E. C. Whitman, "Defense conversion in marine technology," *Sea Technology*, vol. 35, no. 11, pp. 21–25, 1994.

[4] A. Mansour and I. Leblond, "Ecosystem monitoring and port surveillance systems," *Advances in Applied Mechanics*, vol. 2, pp. 91–110, 2013.

[5] C. L. Pekeris, "Theory of propagation of explosive sound in shallow water," *Geological Society of America Memoirs*, vol. 27, no. 1, pp. 1–116, 1948.

[6] S. A. Stotts and R. A. Koch, "A two-way coupled mode formalism that satisfies energy conservation for impedance boundaries in underwater acoustics," *Journal of the Acoustical Society of America*, vol. 138, pp. 1281–1298, 2015.

[7] H. Tu, Y. Wang, W. Liu et al., "A Chebyshev spectral method for normal mode and parabolic equation models in underwater acoustics," *Mathematical Problems in Engineering*, vol. 2020, Article ID 7461314, , 2020.

[8] T. A. Möller, E. Haines, N. Hoffman et al., *Real-Time Rendering*, CRC Press, Boca Raton, FL, USA, 4th edition, 2018.

[9] F. B. Jensen, W. A. Kuperman, M. B. Porter et al., *Computational Ocean Acoustics*, Springer, Berlin, Germany, 2nd edition, 2011.

[10] J. Ianniello, *A MATLAB Version of the KRAKEN Normal Mode Code*, Naval Undersea Warfare Center Technical Memorandum, Newport, RI, USA, 1994.

[11] Unity User Manual 2018.3, Writing post-processing effects, available: https://docs.unity3d.com/2018.3/Documentation/Manual/PostProcessingWritingEffects.html.

[12] Github, *Unity-Technologies/PostProcessing*, Github, San Francisco, CA, USA, 2020, https://github.com/Unity-Technologies/PostProcessing/tree/v2.

[13] P. C. Etter, "Advanced applications for underwater acoustic modeling," *Advances in Acoustics and Vibration*, vol. 2012, Article ID 214839, , 2012.

[14] P. C. Etter, "Recent advances in underwater acoustic modelling and simulation," *Journal of Sound and Vibration*, vol. 240, no. 2, pp. 351–383, 2001.

[15] W. B. Langdon, "Graphics processing units and genetic programming: an overview," *Soft Computing*, vol. 15, no. 8, pp. 1657–1669, 2011.

[16] K. E. Taylor, "Summarizing multiple aspects of model performance in a single diagram," *Journal of Geophysical Research: Atmospheres*, vol. 106, no. D7, pp. 7183–7192, 2001.